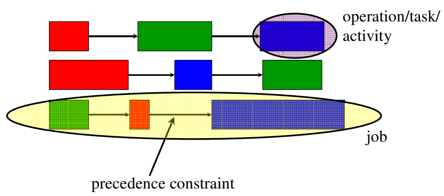
JOB SHOP SCHEDULING

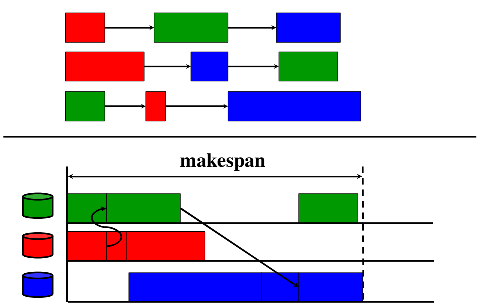
Job Shop Scheduling

- Problem with m machines and n jobs
- Each job visits some or all of the machines
  - Only once (or multiple times if ***recirculation*** is allowed)
- Each customer order is unique and of small batches
  - Wafer fabrication in semiconductor industry
  - Hospital (patients are ***jobs*** or ***machines***?)
- Special case of project scheduling with workforce constraints
- **Very difficult to solve!** (NP-hard)

Job Shop Scheduling



Job Shop Scheduling



Classic scheduling theory

- Look at a specific machine environment with a specific objective
- Analyze to prove an optimal policy or to show that no simple optimal policy exists
- *Thousands of problems have been studied in detail with mathematical proofs!*

Single machine and parallel machines

- Single machine is the simplest models in job shop.
- Parallel machine models are equivalent to flexible flow shop with a single work center.
- These problems can be solved by heuristics (**dispatching rules**).
- Sometimes heuristics can be proved to be optimal for simple cases

Dispatching rules

- Dispatch rule can be **static** or **dynamic**.
- One machine problems (WSPT, EDD, MS, ATC)
- Parallel machines (LPT)
- *Prioritize all waiting jobs*
  - *job attributes*
  - *machine attributes*
  - *current time*
- *Whenever a machine becomes free: select the job with the highest priority*

Release/due date related

- Earliest release date first (ERD) rule
  - variance in throughput times
- Earliest due date first (EDD) rule
  - maximum lateness
- Minimum slack first (MS) rule

$$\max\left(d_j - p_j - t, 0\right)$$

  - maximum lateness
- Apparent tardiness cost first (ATC)
  - maximum total weighted lateness

Processing time related

- Longest Processing Time first (LPT) rule
  - balance load on parallel machines
  - makespan
- Shortest Processing Time first (SPT) rule
  - sum of completion times
  - WIP
- Weighted Shortest Processing Time first (WSPT) rule
- Critical Path (CP) rule
  - precedence constraints
  - makespan

Discussion

- Very simple to implement
- Optimal for special cases
- Only focus on one objective
- Combine several dispatching rules:

Composite Dispatching Rules

Single machine models & WSPT

- n jobs with p_j, r_j and d_j.
- *Total weighted completion time* should be minimized:

$$\sum w_j C_j$$

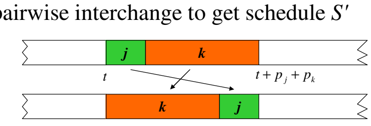
- **Solution**: *Weighted Shortest Processing Time* (WSPT) first is optimal.
  - Schedules jobs in decreasing order of w_j/p_j.
- **SPT** rule starts with job with the shortest p_j, moves on to job with second shortest p_j, and so on.

Proof

- Suppose it is not true and schedule S is optimal.
- Then there are two adjacent jobs, say job j followed by job k such that

$$\frac{w_j}{p_j} < \frac{w_k}{p_k}$$

- Do a pairwise interchange to get schedule S'



Proof

The weighted completion time of the two jobs under S is

$$(t+p_j)w_j + (t+p_j+p_k)w_k$$

The weighted completion time of the two jobs under S' is

$$(t+p_k)w_k + (t+p_j+p_k)w_j$$

Then:

$$\begin{aligned}(t+p_j)w_j + (t+p_j+p_k)w_k &= (t+p_j)w_j + p_j w_k + (t+p_k)w_k \\ &> (t+p_j)w_j + p_k w_j + (t+p_k)w_k \\ &= (t+p_k)w_k + (t+p_k+p_j)w_j\end{aligned}$$

Contradicting that S is optimal.

Single machine models & EDD

- $r_j = 0$
- Each job has its own d_j
- **Objective**: minimize lateness L_{max}
- **Earliest Due Date** (EDD) results in optimal schedule
- Order operations in increasing order of d_j

Types of dispatching rules

- WSPT and EDD are static.
- **Static** – basis for ordering operations does not change based on scheduling decisions
  - All operations can be sorted once
- **Dynamic** – scheduling decisions change the order of remaining operations
  - Need to resort operations in queue (potentially) after every decision

Single machine & Minimum Slack

- $r_j = 0$
- **Objective**: minimize lateness L_{max}
- **Minimum Slack** (MS) orders operations at time t in descending order of:
  - $\max(d_j - p_j - t, 0)$
- *MS does not guarantee optimal schedule!*

Composite rule

Two good heuristics:

- Weighted Shorted Processing Time (WSPT)
  - optimal with due dates zero
- Minimum Slack (MS)
  - Optimal when due dates are "spread out"
- **Any real problem is somewhere in between**
- Combine the characteristics of these rules into one composite dispatching rule.

Composite Dispatching Rule

- One-machine, $r_j = 0$
- **Objective**: minimize weighted tardiness $\sum w_j T_j$
- **Apparent Tardiness Cost** (ATC) rule orders operations in descending order (K is a parameter):

$$I_j(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{K\bar{p}}\right)$$

WSPT

MS

Mean p_j

Special cases

- If K is very large:
  - ATC reduces to WSPT
- If K is very small and no overdue jobs:
  - ATC reduces to MS
- If K is very small and overdue jobs:
  - ATC reduces to WSPT applied to overdue jobs

Choosing K

- Value of K determined empirically
- Related to the *due date tightness* factor

$$\tau = 1 - \frac{\bar{d}}{C_{\max}}$$

and the *due date range* factor

$$R = \frac{d_{\max} - d_{\min}}{C_{\max}}$$

Choosing K

- Usually $1.5 \leq K \leq 4.5$
- Rules of thumb:
  - Fix $K = 2$ for single machine or flow shop.
  - Fix $K = 3$ for dynamic job shops.
- Adjusted to reduce weighted tardiness cost in extremely slack or congested job shops
- Statistical analysis/empirical experience

Jobs with different release dates

- One-machine problem with different r_j
- **Objective**: minimize lateness $L_{\max}$

- Problem is NP-hard
- Possible algorithms to solve the problem
  - **Branch-and-bound** (see Appendix B of Pinedo's book)
  - **Dynamic programming**

Parallel machines

- A set of m machines in parallel is available.
- **Objective**: minimize makespan $C_{\max}$

- **Longest Processing Time** (LPT) first
  - pick operations in descending order of processing time
- LPT balances the loads of the machines (why?).
- LPT does **not** guarantee optimality.

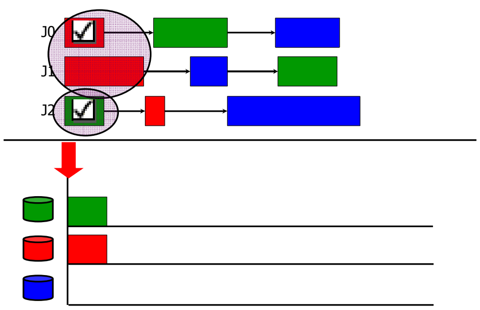
Parallel machines

- **Objective**: minimize completion time $\sum C_j$
- SPT assures optimality, even when preemptions are allowed.

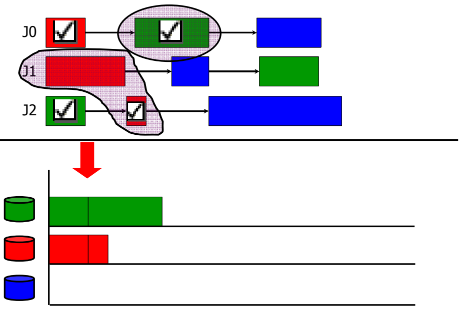
- **Objective**: minimize ***weighted*** completion time $\sum w_j C_j$
- WSPT does **not** assures optimality.

- **Objective**: minimize total weighted tardiness $\sum w_j T_j$
- This more general problem is even harder. ATC can be applied, but solutions can be poor.

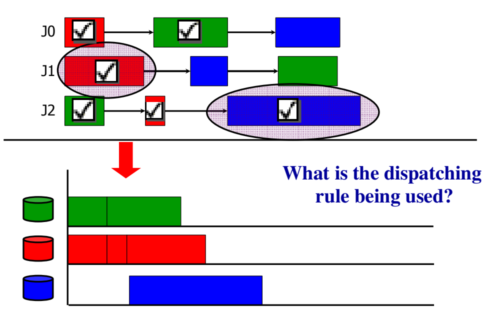
Dispatching rules for Job Shop



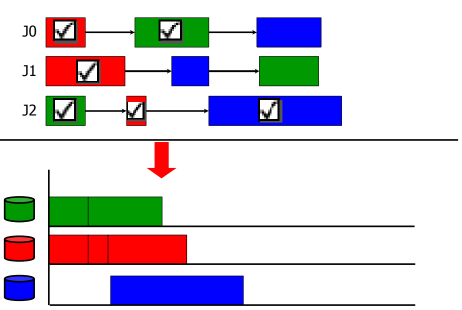
Dispatching rules for Job Shop



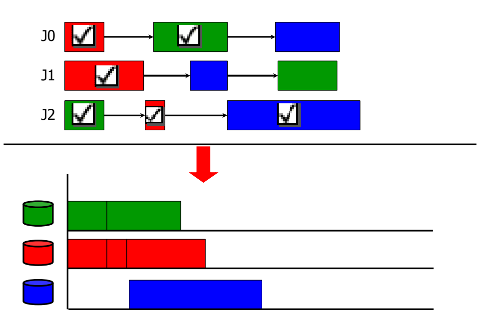
Dispatching rules for Job Shop



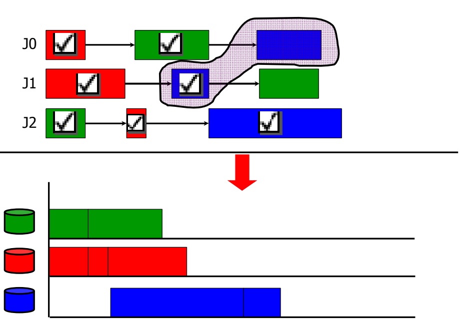
Dispatching rules for Job Shop



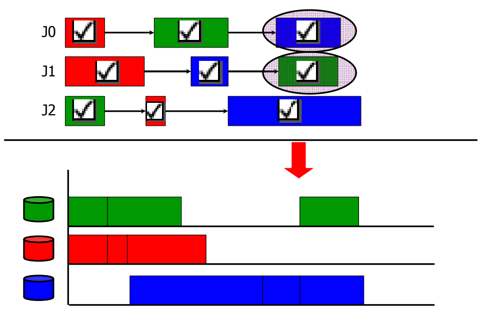
Dispatching rules for Job Shop



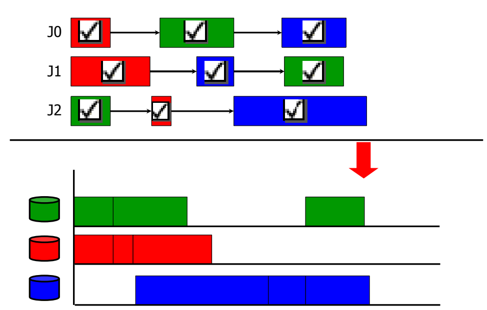
Dispatching rules for Job Shop



Dispatching rules for Job Shop



Dispatching rules for Job Shop



JSP and Mathematical Programming

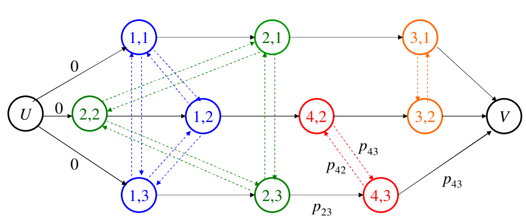
- Job shop with n jobs and m machines.
- Each job visits some machines in a given order *without* recirculation.
- Processing of job j in machine i is operation (i, j) with duration p_{ij}, and $(i, j) \in N$ nodes.
- **Objective**: minimize makespan $C_{\max}$
- Problem can be represented in a **disjunctive graph**.

JSP and Mathematical Programming

- Direct graph $G = (N, A, B)$ with a set of N operations.
- Arcs A - **conjunctive** arcs represent the precedence relationships between processing operations of a job.
- Arcs B - **disjunctive** arcs connect two operations which belong to two different jobs, that are to be processed on the same machine.
- Disjunctive arcs form n **cliques** (in a clique any two nodes are connected to one another).
- Source U and sink V.

Disjunctive graph

Example of a job shop problem: 4 machines and 3 jobs



Disjunctive graph

- **Feasible schedule** – selection of one disjunctive arc from each pair. *Each selection of arcs within a clique must be acyclic*.
- Let D be a subset of selected disjunctive arcs.
- Makespan of a feasible schedule is the longest path in $G(D)$ from the source U to the sink V.
- The problem is thus minimizing the longest (*critical*) path.

Disjunctive programming

- Based on the disjunctive graph.
- Let y_{ij} be the starting time of operation (i, j) (operation of job j in machine i)
- N – set of all operations
- A – set of all conjunctive constraints
- B – set of all disjunctive constraints

Disjunctive programming formulation

minimize $C_{\max}$

subject to

$y_{hj} - y_{ij} \geq p_{ij}$	for all $(i, j) \to (h, j) \in A$
$C_{\max} - y_{ij} \geq p_{ij}$	for all $(i, j) \in N$
$y_{ij} - y_{ik} \geq p_{ik}$ or $y_{ik} - y_{ij} \geq p_{ij}$	for all (i,k) and (i, j)
$y_{ij} \geq 0$	for all $(i, j) \in N$

Disjunctive programming formulation

minimize $C_{\max}$

subject to

$y_{hj} - y_{ij} \geq p_{ij}$	for all $(i, j) \to (h, j) \in A$
$C_{\max} - y_{ij} \geq p_{ij}$	for all $(i, j) \in N$
$y_{ij} - y_{ik} \geq p_{ik}$ or $y_{ik} - y_{ij} \geq p_{ij}$	for all (i,k) and (i, j)
$y_{ij} \geq 0$	for all $(i, j) \in N$

An operation cannot start before the previous operation (in the job) ends

Disjunctive programming formulation

minimize $C_{\max}$

subject to

$y_{hj} - y_{ij} \geq p_{ij}$	for all $(i, j) \to (h, j) \in A$
$C_{\max} - y_{ij} \geq p_{ij}$	for all $(i, j) \in N$
$y_{ij} - y_{ik} \geq p_{ik}$ or $y_{ik} - y_{ij} \geq p_{ij}$	for all (i,k) and (i, j)
$y_{ij} \geq 0$	for all $(i, j) \in N$

All operations must end before makespan

Disjunctive programming formulation

minimize $C_{\max}$

subject to

$y_{hj} - y_{ij} \geq p_{ij}$	for all $(i, j) \to (h, j) \in A$
$C_{\max} - y_{ij} \geq p_{ij}$	for all $(i, j) \in N$
$y_{ij} - y_{ik} \geq p_{ik}$ or $y_{ik} - y_{ij} \geq p_{ij}$	for all (i,k) and (i, j)
$y_{ij} \geq 0$	for all $(i, j) \in N$

One disjunctive arc must be chosen

Disjunctive programming formulation

minimize $C_{\max}$

subject to

$y_{hj} - y_{ij} \geq p_{ij}$	for all $(i, j) \to (h, j) \in A$
$C_{\max} - y_{ij} \geq p_{ij}$	for all $(i, j) \in N$
$y_{ij} - y_{ik} \geq p_{ik}$ or $y_{ik} - y_{ij} \geq p_{ij}$	for all (i,k) and (i, j)
$y_{ij} \geq 0$	for all $(i, j) \in N$

Start times cannot be negative

Solution Methods

- Exact solution
  - Branch & Bound
  - 20 machines and 20 jobs
- Dispatching rules (16+)
  - Shifting Bottleneck
- Search heuristics
  - Tabu search, Simulated Annealing, Genetic Algorithms, etc.

Shifting Bottleneck Heuristic

- Minimize makespan in a job shop
- Let M denote the set of machines
- Let $M_0 \subseteq M$ be machines for which disjunctive arcs have been selected
- **Basic idea**:
  - Select a machine in $M - M_0$ to be included in M_0
  - Sequence the operations on this machine

Shifting Bottleneck Algorithm

Step 1: *Set the initial conditions*

- Set $M_0 = \varnothing$. Graph G is the graph with all the conjunctive arcs and no disjunctive arcs.
- Set $C_{\max}(M_0)$ equal to the longest path in graph G.

Step 2: *Analysis of the machines still to be scheduled*

- For each machine i in $M - M_0$: formulate a single machine problem with all operations subject to release dates and due dates. Release date is the longest path in G from the source to the node. Due date is the longest path in G from the node to the sink and subtracting p_{ij}.
- Minimize $L_{\max}$ in each machine.

Shifting Bottleneck Algorithm

Step 3: *Bottleneck selection*

- The machine with the highest cost is designated the bottleneck.
- Insert all the corresponding disjunctive arcs in graph G.
- Insert machine which is the bottleneck in M_0.

Step 4: *Resequencing all machines scheduled earlier*

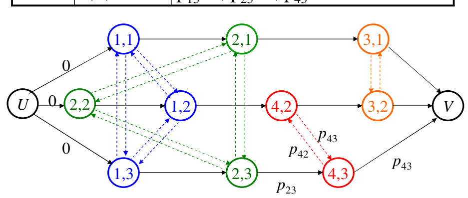
- Find the sequence that minimized the cost and insert the corresponding disjunctive arcs in graph G.

Step 5. *Stopping condition*

- If all machines are scheduled $(M_0 = M)$ then STOP, else go to Step 2.

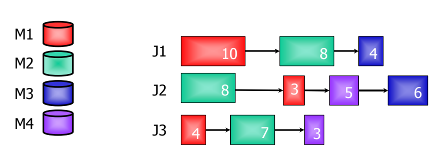
Example 5.4.2 (p. 89)

Jobs	Machines	Processing times
1	1,2,3	p_{11}=10, p_{21}=8, p_{31}=4
2	2,1,4,3	p_{22}=8, p_{12}=3, p_{42}=5 , p_{32}=6
3	1,2,4	p_{13}=4, p_{23}=7, p_{43}=3



Example 5.4.2 (p. 89)

- Other form of presenting processing times:



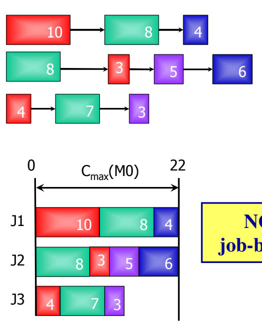
Inicialization

- M = set of all machines
- M_0 = set of "already scheduled" machines
  - Initially M0 is empty

Step 1

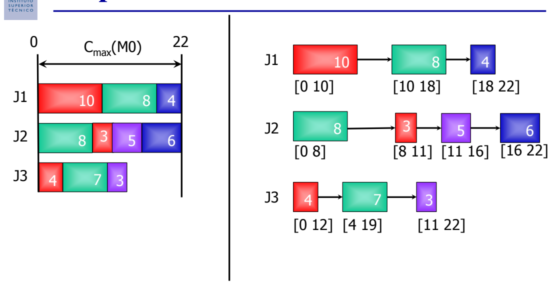
- Find release date and due date of each operation
- Remove all sequence constraints among activities in $M - M_0$, use CPM to find critical path and minimize start time, maximize ending time for each activity
  - Since M_0 is initially empty, only conjunctive arcs appear in the graph.

Step 1: Find $C_{max}(M_0)$



NOTE: This is a job-based Gantt chart

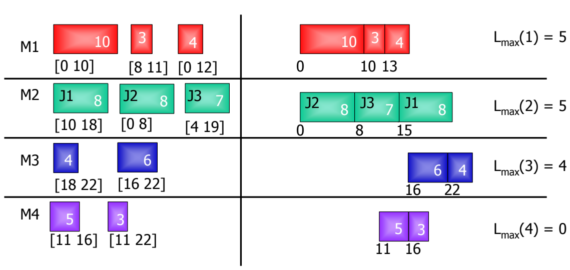
Step 1: Find release and due dates



Step 2: Find optimal 1-machine schedules

- Using release and due dates, min. L_{max}

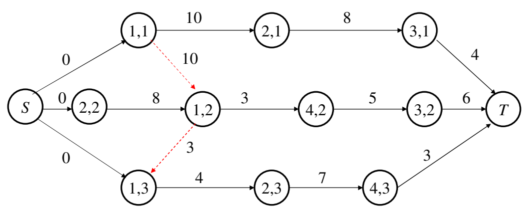
$L_j = C_j - d_j$
$L_{max} = \max(L_j)$



Step 3: Add machine to M_0

- Pick machine with highest L_{max}
- Use sequence found in Step 2
  - $L_{max}(1) = L_{max}(2) = 5$
  - Arbitrarily choose to add machine 1

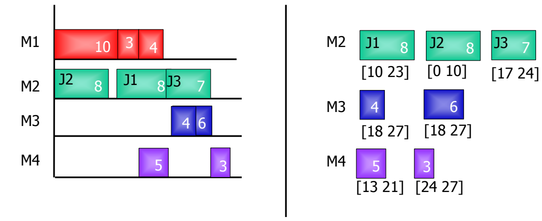
Graph after iteration 1



$C_{max}(\{1\}) = C_{max}(\varnothing) + L_{max}(1) = 22 + 5 = 27$

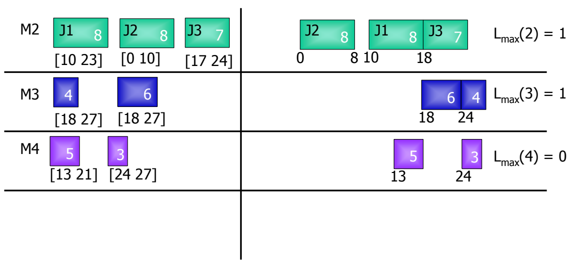
Iteration 2, Step 1

- $M_0 = \{1\}$
- $C_{\max}(M_0) = 27$, find release and due dates



Iteration 2, Step 2

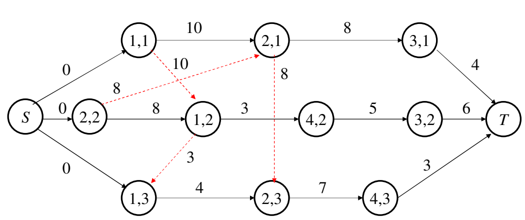
- Using release and due dates, min. $L_{\max}$



Iteration 2, Step 3

- Pick machine with highest $L_{\max}$
- Use sequence found in Step 2
  - $L_{\max}(2) = L_{\max}(3) = 1$
  - Arbitrarily choose to add machine 2

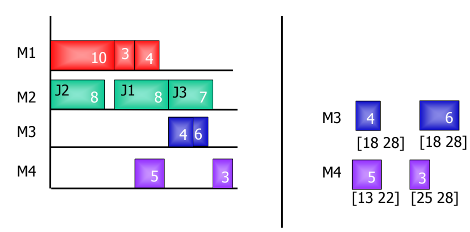
Graph after iteration 2



$C_{max}(\{1,2\}) = C_{max}(\{1\}) + L_{max}(2) = 27 + 1 = 28$

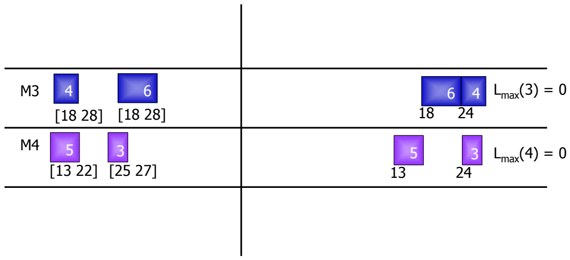
Iteration 3, Step 1

- $M_0 = \{1, 2\}$
- $C_{\max}(M_0) = 28$, find release and due dates



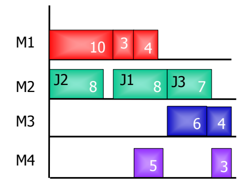
Iteration 3, Step 2

- Using release and due dates, min. L_{max}

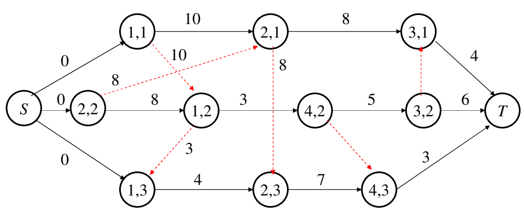


Iteration 3, Step 2

- $L_{max}(3) = L_{max}(4) = 0$
- Thus, a final schedule was found in Step 2



Final graph



Discussion

- Very effective
  - Relatively fast
  - Good solutions
  - More general Job Shop problems can be solved as well
- *'Just a heuristic'*
  - No guarantee of optimum

Branch-and-Bound

- Represent problem as an **Integer Programming** (IP) problem
  - Sequence of every pair of operations is a 0-1 variable
- Use **Branch-and-Bound** (B&B) to find solution
- Will find optimal solution (if given enough time!)

Constraint Programming

- B&B (but not IP) plus inference
- Every time you branch, use specialized algorithms to find other decisions that must be true
- May also use sophisticated branching heuristics
- May need to backtrack (if the heuristic decision has made a mistake)
- Also will find optimal if enough time is given.

LEKIN

- On disk with book
- Generic job shop scheduling system
- User friendly windows environment
- C++ object oriented design
- Can add own routines
- *Look at slides from University of Nottingham coming with book!* or:

  http://www.cs.nott.ac.uk/~sxp/

LEKIN